系统调用层的操作系统安全增强

高 微 卿斯汉 崔永祯

(中国科学院软件研究所信息安全技术工程研究中心 北京100080)

摘 要 在操作系统安全内核的实现方式上,一种主流的做法是直接改造原有系统的实现代码,加入安全机制。其中较为常见的是以LINUX开放源代码为基础进行开发,直接对内核进行修改。在发现这种做法优点的同时,我们也必须意识到它也存在一定的缺点,比如兼容性和移植性的问题。本文提出了安全虚拟机(SVM)这一概念,阐述了从系统调用层对操作系统安全实施增强的技术,并根据此策略设计了一个能够达到C2级安全保护的模型,重点突出了设计思想和实现考虑。

关键词 安全虚拟机,安全策略,可信计算基,访问控制

Operating System Security Enhancement at the System Call Level

GAO Wei QING Si-Han CUI Yong-Zhen

(Engineering and Research Center for Information Security Technology, Institute of Software, CAS, Beijing 100080)

Abstract The one main method of implementing security kernel is, directly modifying the original system's source code, to add security mechanism in operating system. That is to say, implementation mechanism of TCB (Trusted Computing Base)should be embedded in kernel source code. As we know the advantages of this method, we must be aware that some shortcomes exist, such as compatibility and portability. In this paper we put forward a security virtual machine (SVM)concept, discuss the technology of operating system security enhancement at the system call level, and design a security model based on this concept, which can reach the C2 level security protection standard, emphasizing on the design idea and implementation advisement.

Keywords Security virtual machine, Security policy, Trusted computing base, Access control

1 引言

随着IT业的迅猛发展,信息系统安全的重要性已不言而喻。目前国内基本上都是利用国外的技术甚至是部分源代码,根据市场需要自己组合成的操作系统,这种操作系统不具有我们的自主知识产权。以Linux为代表的国际自由软件的发展为我国发展具有自主版权的系统软件提供了良好的机遇。然而就安全性而言,非开放源代码的操作系统是个黑盒子,而一个源代码公开的系统更像是一个玻璃盒子,这恐怕也无法让人安心。

对于主流的操作系统安全机制,即以Linux开放源代码为基础进行开发,将可信计算基(TCB)[1]的实现机制嵌入到内核代码中,其优点是系统实现紧凑而高效,应用程序无法绕过嵌入内核中的引用监视器,因而安全机制不可旁路。但也存在着一些缺点,最突出的便是整个系统的兼容性要受到一定程度的影响。同时,由于大多数应用程序是在原来的未加入安全机制的环境下开发的,因而要在实现了安全策略的新平台上稳定运行会或多或少地存在一些问题,这些问题大多是由兼容性引起的。另外,由于安全机制是在内核中固定实现的,因此不具有可移植性。在某类操作系统上实现的安全机制无法应用在另一类系统上,存在着重复开发的问题。即使在同一类操作系统上,当出现新版本的内核时,也要对原有的代码进行一定的改造之后才能被新的系统采用。这就带来了各类安全操作系统安全机制的实现方式、与应用程序的接口和使用管理方法等多方面互不统一、使用不便、难以扩充和推广等方面的问题。

因此,研究和实现支持动态、灵活、可移植的安全策略的安全模型是近年来信息安全领域内的一个热点问题。安全虚拟机(Security Virtual Machine)的模型就是在这样的背景下为解决当前在安全操作系统研究与开发方面存在的诸类问题而实现的新型安全内核。它抛弃了传统的做法,不对原有系统进行代码级的改造,而是对原有系统进行安全封装,在此基础上建立起一个具有安全机制的虚拟机,为用户提供一个兼具安全性、可用性和兼容性的系统服务平台。

2 系统结构描述

安全模块是整个安全系统平台的中心,它对外表现为一个透明的安全虚拟机(SVM),也可称之为TCB(可信计算基)虚拟机。其在整个系统平台中所处的位置以及与下层原有操作系统和上层应用服务平台之间的关系由图1所示。

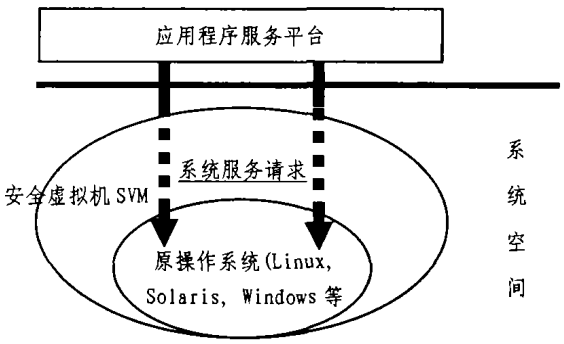

图1 安全虚拟机(SVM)

从图1可以看出,SVM是对原有操作系统的一个封装,对

高 微 硕士生,研究方向为操作系统安全;卿斯汉 研究员,博士生导师;崔永祯 硕士生。

上层应用程序的系统请求提供了一个透明的服务平台。这样，应用程序觉察不到虚拟机的存在，但却受到安全机制的制约。同时，安全虚拟机本身具有良好的抗攻击性能，因此实际上将原操作系统与外界环境安全地隔离了，让它在一个十分安全的环境下运行，提高了整个系统的安全性和稳定性。

2.1 系统的网络环境

如图2所示，基于易操作性的考虑，SVM 机系统采用集中管理的方式进行配置，管理平台与被管理的节点代理(Agent)之间采用加密信道进行通信。此管理平台可以与被管理的所有节点代理(Agent)在同一个局域网内，也可以远程进行。

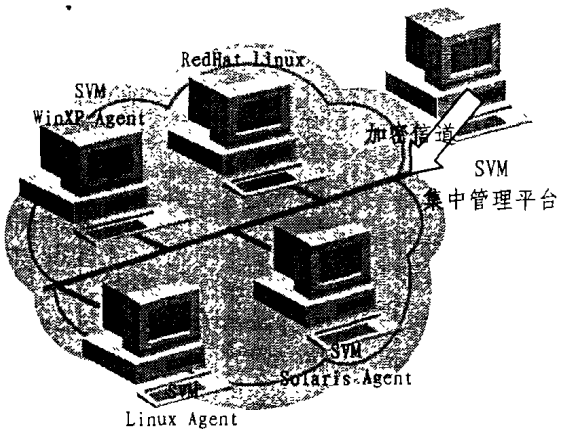

图2 系统的网络环境

2.2 系统的结构设计

SVM 系统由多个子模块构成，我们将功能进行细化。每个子模块通过严格定义的接口进行通信，互相作用实现整个系统的安全功能。从逻辑上来说，可以将安全内核拆分为若干个主要的功能体，它们之间的关系如图3所示。

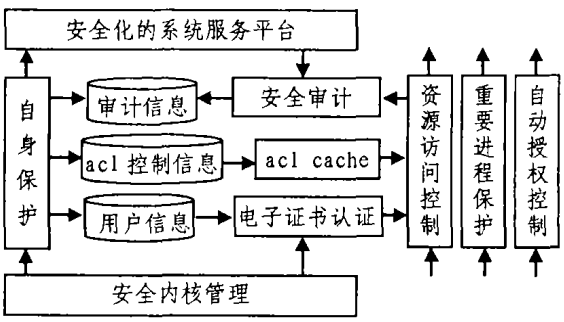

图3 系统的结构关系

2.3 系统特点

与采用传统方式实现的安全内核相比，从系统实现上来说(不考虑安全机制)，SVM 具有如下几大特点：

·兼容性强 SVM 不对原有系统的代码进行任何修改，而是在系统运行时从应用层载入相应的安全机制，对原有系统进行封装后提供一个具有安全内核的虚拟机。SVM 对应用程序透明，因而原来所有的系统服务仍然存在。以应用程序的角度来看，觉察不出系统的任何改变，但它受到虚拟机内安全引用监视器的控制，对所有对象的访问都受到安全机制的检查，并且这些安全机制是不可旁路的。这就带来高安全性且对原有系统的可用性不产生影响。

·系统可移植 SVM 实现了基于模块机制的内核封装方法；同时，许多操作系统都采用了模块化的设计与实现方式，因而都能用 SVM 对它们进行安全封装。这些操作系统涵盖了目前市场上常见和流行的服务器系统，如开放源码的 Linux，SUN 的 Solaris，HP 的 HPUX，IBM 的 AIX，以及微软的 Windows 等。这是一种十分通用的系统安全实现方式，因此可移植性好，符合现代软件设计方法学。

·易于扩充 具有安全内核的虚拟机是对原有系统的抽象，我们可以在此基础上加入任何想要的安全功能或机制。重要的是所有安全功能的实现都与底层的系统无直接关系，使用虚拟机提供的对底层的抽象接口即可。

·使用方便 安全操作系统大都给人以难以使用、界面不友好、配置繁琐、维护困难、只有专业人士才能使用等等印象。这大大影响了安全操作系统与推广，也不便于真正发挥出其原有的安全功能。SVM 在实现安全机制的同时也考虑到了上述问题，因而其所有配置工具都采用了良好的用户界面。其一大优点是实现了基于 Agent 的网络配置方式，使得系统管理员可以在 Windows 环境下对任何一台安装了 SVM 的机器进行远程配置。这大大方便了用户，其配置和使用界面也是十分友好的，即使没有受过专业训练的用户也能使用。

·简单实用 由于不需要对原有系统进行修改和扩充，因此在提供同级别安全功能的基础上大大减少了工作量和开发时间，也相应地减少了开发成本，有利于产品化推广。

以上所说的都是 SVM 的优点，但也同时存在着一些不利因素。最主要的就是对系统运行性能的影响。由于不对原系统的代码进行修改，因而安全机制不能被嵌入内核，抽象出来的虚拟机会对某些系统服务的运行效率产生负面影响。这些影响主要集中在对客体访问上下文的建立阶段，如打开文件、访问文件属性等。为减少这些负面因素，我们主要采用了缓存机制，在虚拟机内建立了多个 LRU 缓存器(cache)。其中，用户可以从节点数和内存量两个方面控制最重要的 ACL 缓存占用系统资源的大小，根据实际需要进行调整。实践证明，对缓存的使用大大降低了安全引用监视器对系统性能带来的负面影响。在正常使用过程中，这些缓存的命中率大都在95%以上。在大多数计算环境下，由安全机制带来的迟滞现象是可以忽略不记的。在具有频繁的对象访问操作的环境下，对运行效率的最大影响也不会达到受到重视的程度。在计算部件快速发展和越来越廉价的今天，这不是一个大问题。

3 安全功能

作为一个安全内核(Secure Kernel)，SVM 从多个方面完整地保护整个系统的信息资源。目前 SVM 的安全策略是基于自主访问控制(DAC)即客体 ACL 列表控制[2,3]，并且开发和实现了许多新的安全策略实施机制，能够达到美国国防部国家计算机安全中心(NCSC)制定的 C2级别的安全保护等级(TCSEC)标准。同时，其开放性和易于扩充的系统结构便于实现更多的安全机制，可以达到更高的安全保护等级。

目前 SVM 模型具有的安全功能主要包括以下几个方面：

·ACL 客体访问控制机制 允许对客体设置多达12类的访问控制权限；每个用户(组)的访问权限包括拒绝/允许两种(拒绝优先于允许)；支持访问控制权限的继承关系；所有的 ACL 信息都存在一个安全数据库中，受到安全内核的封装和保护，只允许系统管理员(Administrator)访问。

·系统服务控制机制 系统服务接口是应用层和内核之间联系的重要通道。安全内核截取了原系统中与安全相关的几十个系统调用，增加其安全性并加入必要的访问控制机制，从而任何来自应用层的系统服务请求都会经过安全虚拟机内

安全引用监视器的过滤，阻止未授权或不安全的请求，实现预设的安全策略。

·基于电子证书的身份认证　系统向授权用户颁发具有系统管理员签名的电子证书，并且采用操作系统内核级的证书认证方式，用户只能通过该证书进行内核级的认证和登录。证书格式类似于 X.509，支持 RSA 数字签名算法，MD5等 hash 算法，以及多种证书加密算法，如 AES、QC5等。系统向用户提供自行加载其他算法的接口。

·用户分组　可以依据实际操作环境、业务、责任等建立信息使用策略，对用户进行分组，以组为单位实施对客体的访问控制安全策略[5]，并可以对系统管理员实施角色控制。

·安全内核自身保护　SVM 在启动阶段首先完成对原系统的安全封装(Kernel Seal)，然后保护好系统目录和配置资源并将自身隐藏(Kernel Conceal)，因此普通用户或进程无法探知安全内核的存在，也就无法实施任何有效的攻击了。系统重启、关闭、模块加载等操作对其他用户都无效(系统管理员除外)，因此黑客攻击者或普通用户无法结束安全内核，引起安全功能的失效。同时，系统因突然断电等突发事件重启时，会继续保持原来的功能，通过维持安全设置解决系统重启以后的安全漏洞。

·防止非法结束重要进程　系统管理员可以对一些重要的系统服务进程，如网页服务、数据库服务等进行保护，防止非法结束(Anti-Kill)，保证服务的正常运行。

·自动进程授权　系统管理员可以对某些可信的或频繁使用的应用程序设置自动授权功能(Auto-ID)，从而使该程序在启动后自动以某授权用户的身份运行。通过使该进程只拥有某些特定的权限，将其限制在一个相对独立的执行域内进行操作，提高服务平台的安全和稳定性。

·内核级的审计功能　在内核层记录安全管理员及用户的所有行为确保安全。审计记录的产生以系统调用为单位，内容包括主、客体，发生时间，事件类型，结果等。同时，在应用层提供了方便的查询工具，提供多样化的查询条件。基于系统调用的事件审计方式是可配置的，而且为实现基于系统调用事件序列的入侵监测系统[4]建立了基础。

·网络化的服务器管理方式　以 Manager 和 Agent 的方式提供对单一或多数服务器的网络化统一管理，提高安全管理效率。该管理内容包括所有的安全功能、安全策略和电子证书的发放、用户/组成员的建立和撤销、审计信息的查阅等。Manager 以 Windows 为运行平台，提供电子证书认证系统，并且用临时产生的对称密钥对双方的通信过程进行加密处理。整个界面的设计尽量人性化，可操作性要高，以消除复杂繁琐的操作和维护动作对用户产生的不利影响，充分发挥出整个系统的功能。

SVM 对原操作系统进行了封装，其主要过程就是截取了原系统的系统调用，代之以虚拟机的安全系统调用。这种机制应用层是无法旁路的，而且对应用层来说也完全透明，致使应用服务最终得以在新的安全平台上运行。对 ACL 的设置形成了安全策略，而安全系统调用保证了安全策略的实施，实现了安全功能。

我们不需要对所有的系统调用都进行截取和改造，只要关注和系统安全有关的部分即可。根据服务类别和实现的功能，对它们进行分类、总结和编号，结合基本权限判断哪些权限需要在哪些系统调用中进行控制。这其中的大部分是可以设置的，可以在应用层决定对哪些系统调用进行截取。除此之外，有些系统调用是必须被强制性截取的，以保证安全功能的实现。

4　性能分析

上面已经谈到，在安全策略实施代码未嵌入原系统代码的情况下，安全机制实施部分会对原系统的运行效率产生一定的影响。这主要表现在主体身份和客体 ACL 列表的获得、权限的判断、安全会话的建立等各个阶段。因为客体 ACL 存在物理存储媒介中，每一次检查过程都要从物理媒介中取出 ACL 列表进行判断，所以无数次频繁的读文件操作大大增加了运行时间。这些影响主要集中在文件的打开操作和一些对文件属性的操作上。

解决这个问题首先想到的自然是加入缓存，根据最近访问优先的原则，将业已获得的 ACL 列表继续保存在内存中，以避免下次用到时再到磁盘中去读。应用程序的运行环境大都具有局部性的特点，所以经过一段时间，大多数的 ACL 可以直接从缓存中得到，而不需要每次都启动磁盘操作。这一点可以从正常运行环境下的缓存命中率得到证明。在将缓存设置为10,000个节点大小，耗费空间不超过1MB 的情况下，命中率在98%以上。至于实现缓存机制的算法和数据结构目前已非常成熟，有几种结构可供选择：平衡二叉树(avl tree)、红黑树(red-black tree)和分裂树(splay tree)。根据理论计算，它们的查找时间都为 $O(\log n)$ 的数量级(n 为树中节点的个数)。但它们的实现机制和实现复杂度不同，具体用在 ACL 缓存上，可能会有所差别。

经过实践检验，证明了加入 ACL 缓存是对的，可以用不大的系统开销获得极大的速度提升。在以频繁的文件操作为主的运行环境中，内核空间的运行时间增加量大大下降了，最大不会超过20%～30%。在常见的计算环境中，运行效率几乎不受影响。

5　内核封装和保护

SVM 在启动后做的第一件事就是对原系统进行封装保护，然后读入一系列的配置文件，建立好 SVM 的运行环境，实现自身的隐藏和保护后才最后进入服务状态。对原系统内系统调用的截取是实现内核封装的主要手段，其次包括对模块装/卸载的控制和系统重启的控制。对自身的保护包括安全模块的隐藏和对系统配置文件的保护。

5.1　安全模块的隐藏

安全模块载入后必须实现自身的隐藏。完成隐藏后，任何用户都无法获得关于安全模块的任何信息，包括模块名、输出符号表、运行状态等。

根据 Linux 可加载内核模块实现机制的特点，我们通过修改内核模块链表 module_list 的方法实现安全模块的完全隐藏。

5.2　安全模块的自我保护

上面谈到的安全模块的隐藏也是实现自我保护的重要一环。除此之外，必须对系统配置文件加以保护，截取模块系统调用和控制系统重启。

安全模块每次初始化时，都要对重要的系统配置文件进行保护。这通过对系统目录设置 ACL 列表完成。只有系统管理员才能进入这个目录进行操作。由于安全模块在重启时都要重新设置这个目录的 ACL 列表，因此即使原来的 ACL 配置文件丢失了，该文件也会自动重新生成。

SVM 还要控制一些重要的会转变系统服务模式和使系统重启或关闭的命令。

总结　本文通过实例研究，提出了一种从系统调用层对

(下转第193页)

用项目的开发中回答如何表示 agent 的问题提供依据。

确定了表1的四个因素的取值，也就在一定程度上确定了如何表示 agent。但是不同的应用领域具有不同的特点和需求，因此本文提出了一些与领域特点相关的应用原则，如图1所示。

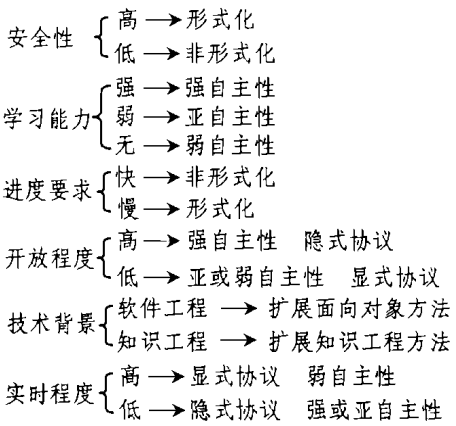

图1 应用原则

在实际系统开放或方法设计中，应综合考虑应用领域的特点，参考本文提出的应用原则，选择适当的方式描述 agent。

总结与展望 本文对 AOSE 的起源，研究现状和典型方法作了简单介绍，首次提出了 agent 的强、亚和弱自主性的定义，讨论了 agent 交互协议的不同描述方式，并根据四个主要因素比较了现有的典型方法，这些既可以帮助软件开发人员对面向 agent 软件开发方法有更深入的了解，可以指导面向 agent 软件开发方法的设计，同时也可指导具体系统的实现，对于 AOSE 和 agent 技术的发展具有积极的意义。

AOSE 的出现只有几年的时间，仍然处于探索阶段，但是发展的势头迅猛，在未来的几年里，还会有更多的面向 agent 的软件开发方法出现。agent 作为一种新的软件开发范型，已经在某些领域获得成功应用，在目前计算机科学领域中，agent 概念的出现频率非常高，这说明 agent 已经不仅仅是一种具有吸引力的概念，其应用价值逐渐获得认可，越来越多的研究人员加入这一领域，因此，我们对于 agent 技术的发展持乐观态度。

参 考 文 献

1 陆汝钤. 知识科学与计算科学. 清华大学出版社，2003
2 樊晓聪. 软件 agent 理论及其方法学的研究：[南京大学博士论文]. 1999
3 Jennings N R. On agent-based software engineering. Artificial Intelligence，2000
4 Towards a practical methodology for agent-oriented software engineering with c++and java
5 Wooldridge M. Agent-based software engineering. 1997
6 Bauer B，Muller J，Odell J. Agent UML：formalism for specifying multiagent interaction. 2000
7 Wagner G. The agent-object-relationship metamodel：Towards a unified conceptual view of state and behavior. 2002
8 Caire C，et al. Agent oriented analysis using MESSAGE/UML. 2001
9 Wooldridge M，Jennings N R，Kinny D. The Gaia Methodology for Agent-Oriented Analysis and Design. In Journal of Autonomous Agents and Multi-Agent Sytems，2000，3(3)：285～312
10 Kinny D，Georgeff M，Rao A. A methodology and modelliing technique for systems of BDI agents. 1996
11 Crnogorac L，Rao A，Ramamohanarao K. Analysis of Inheritance Mechanism in Agent Oriented programming. IJCAI97，1997. 647～652
12 Xu Haiping，Shatz S M. A Framework for Modeling Agent-Oriented software[A]. In：Proc. of the 21th Intl. Conf. on Distributed Computing Systems(ICDCS-21)[C]，2001
13 Köhler M，Rolke H. Towards a Unified Approach for Modeling and verification of Multi Agent Systems，2001
14 Luck M，d'Inverno M. A formal framework for agency and autonomy. In：proc. of the First Intl. Conf. on Multi-Agent Systems (ECMAS-95)，San Francisco，CA，June 1995. 254～260
15 刘璜. 面向 agent 的软件需求分析：[中科院博士论文]. 2000
16 Deloach S A，Wood M F. Sparkman C H. Multiagent systems Engineering. The International Journal of Software Engineering and Knowledge Engineering，2001，11(3)
17 Bauer B. UML Class Diagrams revisited in the context of agent-based systems. In：M. wooldridge，P. Ciancarini，G. Weiss，eds. Proc. of Agent-Oriented Software Engineering(AOSE 01)，number 2222 in LNCS，Montreal，Canada，Springer-Verlag，2001. 1～8
18 Hughet M-P. Agent UML Class Diagrams Revisited. In：Proc. of Agent Technology and Softwarwe Engineering(AgeS). Bernhard Bauer，Klaus Fischer，Jorg Mueller and Bernhard Rumpe(eds)，Erfurt，Germany，Oct. 2002
19 Iglesias C A，et al. Analysis and design of multiagent systems using MAS-CommonKADS. In：AAAI'97Workshop on Agent Theories，Architectures and Languages，Providence，RI，ATAL，1997
20 Iglesias C A，Garijo M，Gonzalez J C. A Survey of Agent-Oriented Methodologies. In：J. P. Muller，M. P. Singh，A. Rao. eds. Intelligent Agents V(ATAL'98)，LNAL 1555，Springer-Verlag，Berlin，Germany，1999. 317～330
21 Wooldridge M，Ciancarini P. Agent-Oriented software Engineering：The state of the Art. In：P. Ciancarini，M. Wooldridge，eds. Agent-Oriented Software Engineering. Springer-Verlag Lecture Notes in AI Volume 1957，Jan. 2001
22 Wooldridge M，Jennings N R. Intelligent agents：Theory and practice[J]. The Knowledge Engineering Review，1995，10(2)：115～152
23 王栩. Agent 通讯系统理论及组织结构研究：[中科院博士论文]. 2000

(上接第178页)

操作系统进行安全增强的概念，系统地分析了此方法的优势和不足之处，并且从设计和实现的角度出发提出了相应的策略。此方法是一个完整的安全体系，它考虑到了系统安全所需的大多方面，诸如访问控制、标识与鉴别、审计、身份认证等。在难以改造内核或对灵活性要求高的安全应用中，本文所描述的策略不失为一上选。

参 考 文 献

1 Bell D E，LaPadula L J. Secure Computer Systems：Mathematical Foundations：[Technical Report M74-244]. The MITRE Corporation，Bedford，Massachusetts，May 1973
2 Sandhu R S. Role-Based Access Control. Advances in Computers，Academic Press，1998，46：237～286
3 Portable Applications Standards Committee of the IEEE Computer Society. Draft Standard for Information Technology-POSIX-Part 1：System Application Program Interface (API)-Amendment：Protection，Audit and Control Interfaces. IEEE Draft P1003.1e，1997
4 Hofmeyr S A，Forrest S，Somayaji A. Intrusion detection using sequences of system calls. Journal of Computer Security，1998，6：151～180